Compiling a Partition-Based Two-Level Formalism

Edmund Grimley-Evans*
University of Cambridge
(St John's College)
Computer Laboratory
Cambridge CB2 3QG, UK
Edmund.Grimley-Evans@cl.cam.ac.uk

George Anton Kiraz†
University of Cambridge
(St John's College)
Computer Laboratory
Cambridge CB2 3QG, UK
George.Kiraz@cl.cam.ac.uk

Stephen G. Pulman
University of Cambridge
Computer Laboratory
Cambridge CB2 3QG, UK
and SRI International, Cambridge
sgp@cam.sri.com

Abstract

This paper describes an algorithm for the compilation of a two (or more) level orthographic or phonological rule notation into finite state transducers. The notation is an alternative to the standard one deriving from Koskenniemi's work: it is believed to have some practical descriptive advantages, and is quite widely used, but has a different interpretation. Efficient interpreters exist for the notation, but until now it has not been clear how to compile to equivalent automata in a transparent way. The present paper shows how to do this, using some of the conceptual tools provided by Kaplan and Kay's regular relations calculus.

1 Introduction

Two-level formalisms based on that introduced by (Koskenniemi, 1983) (see also (Ritchie et al., 1992) and (Kaplan and Kay, 1994)) are widely used in practical NLP systems, and are deservedly regarded as something of a standard. However, there is at least one serious rival two-level notation in existence, developed in response to practical difficulties encountered in writing large-scale morphological descriptions using Koskenniemi's notation. The formalism was first introduced in (Black et al., 1987), was adapted by (Ruessink, 1989), and an extended version of it was proposed for use in the European Commission's ALEP language engineering platform (Pulman, 1991). A further extension to the formalism was described in (Pulman and Hepple, 1993).

The alternative partition formalism was motivated by several perceived practical disadvantages to Koskenniemi's notation. These are detailed more fully in (Black et al., 1987, pp. 13-15), and in (Ritchie et al., 1992, pp. 181-9). In brief: (1) Koskenniemi rules are not easily interpretable (by the grammarian) locally, for the interpretation of 'feasible pairs' depends on other rules in the set. (2) There are frequently interactions between rules: whenever the lexical/surface pair affected by a rule A appears in the context of another rule B, the grammarian must check that its appearance in rule B will not conflict with the requirements of rule A. (3) Contexts may conflict: the same lexical character may obligatorily have multiple realisations in different contexts, but it may be impossible to state the contexts in ways that do not block a desired application. (4) Restriction to single character changes: whenever a change affecting more than one adjacent character occurs, multiple rules must be written. At best this prompts the interaction problem, and at worst can require the rules to be formulated with under-restrictive contexts to avoid mutual blocking. (5) There is no mechanism for relating particular rules to specific classes of morpheme. This has to be achieved indirectly by introducing special abstract triggering characters in lexical representations. This is clumsy, and sometimes descriptively inadequate (Trost, 1990).

Some of these problems can be alleviated by the use of a rule compiler that detects conflicts such as that described in (Karttunen and Beesley, 1992). Others could be overcome by simple extensions to the formalism. But several of these problems arise from the interpretation of Koskenniemi rules: each rule corresponds to a transducer, and the two-level description of a language consists of the intersection of these transducers. Thus somehow or other it must be arranged that every rule accepts every two-level correspondence. We refer to this class of formalisms as 'parallel': every rule, in effect, is applied in parallel at each point in the input.

*Supported by SERC studentship no. 92313384.
†Supported by a Benefactors' Studentship from St John's College.

The partition formalism consists of two types of rules (defined in more detail below) which enforce optional or obligatory changes. The notion of well-formedness is defined via the notion of a 'partition' of a sequence of lexical/surface correspondences. Informally, a partition is a valid analysis if (i) every element of the partition is licensed by an optional rule, and (ii) no element of the partition violates an obligatory rule.

We have found that this formalism has some practical advantages: (1) The rules are relatively independent of each other. (2) Their interpretation is more familiar for linguists: each rule copes with a single correspondence: in general you don't have to worry about all other rules having to be compatible with it. (3) Multiple character changes are permitted (with some restrictions discussed below). (4) A category or term associated with each rule is required to unify with the affected morpheme, allowing for morpho-syntactic effects to be cleanly described. (5) There is a simple and efficient direct interpreter for the rule formalism.

The partition formalism has been implemented in the European Commission's ALEP system for natural language engineering, distributed to over 30 sites. Descriptions of 9 EU languages are being developed. A version has also been implemented within SRI's Core Language Engine (Carter, 1995) and has been used to develop descriptions of English, French, Spanish, Polish, Swedish, and Korean morphology. An N-level extension of the formalism has also been developed by (Kiraz, 1994; Kiraz, 1996b) and used to describe the morphology of Syriac and other Semitic languages, and by (Bowden and Kiraz, 1995) for error detection in nonconcatenative strings. This partition-based two-level formalism is thus a serious rival to the standard Koskenniemi notation.

However, until now, the Koskenniemi notation has had one clear advantage in that it was clear how to compile it into transducers, with all the consequent gains in efficiency and portability and with the ability to construct lexical transducers as in (Karttunen, 1994). This paper sets out to remedy that defect by describing a compilation algorithm for the partition-based two-level notation.

2 Definition of the Formalism

2.1 Formal Definition

We use n tapes, where the first N tapes are lexical and the remaining M are surface, $n = N + M$. In practice, $M = 1$. We write Σ_i for the alphabet of symbols used on tape i, and $\Sigma = (\Sigma_1 \cup \{\epsilon\}) \times ... \times (\Sigma_n \cup \{\epsilon\})$, so that Σ^* is the set of string-tuples representing possible contents of the n tapes. A proper subset of regular n-relations have the property that they are expressible as the Cartesian product of n regular languages, $R = R_1 \times ... \times R_n$; we call such relations 'orthogonal'. (We present our definitions along the lines of (Kaplan and Kay, 1994)).

We use two regular operators: `Intro` and `Sub`. $\texttt{Intro}_S L$ denotes the set of strings in L into which elements of S may be arbitrarily inserted, and $\texttt{Sub}_{A,B} L$ denotes the set of strings in L in which substrings that are in B may be replaced by strings from A. Both operators map regular languages into regular languages, because they can be characterised by regular relations: over the alphabet Σ, $\texttt{Intro}_S = (\texttt{Id}_\Sigma \cup (\{\epsilon\} \times S))^*$, $\texttt{Sub}_{A,B} = (\texttt{Id}_\Sigma \cup (B \times A))^*$, where $\texttt{Id}_L = \{(s, s) \mid s \in L\}$, the identity relation over L.

There are two kinds of two-level rules. The context restriction, or optional, rules, consist of a left context l, a centre c, and a right context r. Surface coercion, or obligatory, rules require the centre to be split into lexical c_l and surface c_s components.

Definition 2.1 A N:M **context restriction (CR) rule** is a triple (l, c, r) where l, c, r are 'orthogonal' regular relations of the form $l = l_1 \times ... \times l_n$, $c = c_1 \times ... \times c_n$, $r = r_1 \times ... \times r_n$. □

Definition 2.2 A N:M **surface coercion (SC) rule** is a quadruple (l, c_l, c_s, r) where l and r are 'orthogonal' regular relations of the form $l = l_1 \times ... \times l_n$, $r = r_1 \times ... \times r_n$, and c_l and c_s are 'orthogonal' regular relations restricting only the lexical and surface tapes, respectively, of the form $c_l = c_1 \times ... \times c_N \times \Sigma^*_{N+1} \times ... \times \Sigma^*_{N+M}$ and $c_s = \Sigma^*_1 \times ... \times \Sigma^*_N \times c_{N+1} \times ... \times c_{N+M}$. □

We usually use the following notation for rules:

LLC – LEX – RLC $\Rightarrow|\Leftarrow|\Leftrightarrow$
LSC – SURF – RSC

where
LLC (left lexical context) = $\langle l_1, \ldots, l_N \rangle$
LEX (lexical form) = $\langle c_1, \ldots, c_N \rangle$
RLC (right lexical context) = $\langle r_1, \ldots, r_N \rangle$
LSC (left surface context) = $\langle l_{N+1}, \ldots, l_{N+M} \rangle$
SURF (surface form) = $\langle c_{N+1}, \ldots, c_{N+M} \rangle$
RSC (right surface context) = $\langle r_{N+1}, \ldots, r_{N+M} \rangle$

Since in practice all the left contexts l_i start with Σ^*_i and all the right contexts r_i end with Σ^*_i, we omit writing it and assume it by default. The operators are: $\Rightarrow$ for CR rules, $\Leftarrow$ for SC rules and $\Leftrightarrow$ for composite rules.

A proposed morphological analysis P is an n-tuple of strings, and the rules are interpreted as

applying to a section of this analysis in context: $P = P_l P_c P_r$ (n-way concatenation of a left context, centre, and right context). Formally:

Definition 2.3 A CR rule (l, c, r) **contextually allows** (P_l, P_c, P_r) iff $P_l \in l$, $P_r \in r$ and $P_c \in c$. □

Definition 2.4 An SC rule (l, c_l, c_r, r) **coercively disallows** (P_l, P_c, P_r) iff $P_l \in l$, $P_r \in r$, $P_c \in c_l$ and $P_c \notin c_s$. □

Definition 2.5 A N:M **two-level grammar** is a pair $(R_\Rightarrow, R_\Leftarrow)$, where $R_\Rightarrow$ is a set of N:M context restriction rules and $R_\Leftarrow$ is a set of N:M surface coercion rules. □

Definition 2.6 A two-level grammar $(R_\Rightarrow, R_\Leftarrow)$ **accepts** the string-tuple P, partitioned as $P_1, ..., P_k$, iff $P = P_1 P_2 ... P_k$ (n-way concatenation) and (1) for each i there is a CR rule $A \in R_\Rightarrow$ such that A contextually allows $(P_1 ... P_{i-1}, P_i, P_{i+1} ... P_k)$ and (2) there are no $i \leq j$ such that there is an SC rule $B \in R_\Leftarrow$ such that B coercively disallows $(P_1 ... P_{i-1}, P_i ... P_{j-1}, P_j ... P_k)$.

There are some alternatives to condition (2):

(2i) there is no i such that there is an SC rule $B \in R_\Leftarrow$ such that B coercively disallows $(P_1 ... P_{i-1}, P_i, P_{i+1} ... P_k)$: this is (2) with the restriction $j = i + 1$; since SC rules can only apply to the partitions P_i, epenthetic rules such as $(\Sigma^* \langle k, k \rangle, \epsilon \times \Sigma_2^*, \Sigma_1^* \times a, \langle k, k \rangle \Sigma^*)$ ('insert an a between lexical and surface ks') can not be enforced: the rule would disallow adjacent $\langle k, k \rangle$s only if they were separated by an empty partition: $...\langle k, k \rangle, \epsilon, \langle k, k \rangle ...$ would be disallowed, but $...\langle k, k \rangle, \langle k, k \rangle ...$ would be accepted.

(2ii) there is no i such that there is an SC rule $B \in R_\Leftarrow$ such that B coercively disallows $(P_1 ... P_{i-1}, P_i, P_{i+1} ... P_k)$ or B coercively disallows $(P_1 ... P_{i-1}, P_i ... P_k)$: this is (2) with the restriction $j = i + 1$ or $j = i$; this allows epenthetic rules to be used but may in certain cases be counterintuitive for the user when insertion rules are used. For example, the rule $(\Sigma^* \langle g, g \rangle, u \times \Sigma_2^*, \Sigma_1^* \times v, \Sigma^*)$ ('change u to v after a g') would not disallow a string-tuple partitioned as $...\langle g, g \rangle, \langle \epsilon, e \rangle, \langle u, u \rangle ...$ – assuming some CR rule allows $\langle \epsilon, e \rangle$.

Earlier versions of the partition formalism could not (in practice) cope with multiple lexical characters in SC rules – see (Carter, 1995, §4.1). This is not the case here.

The following rules illustrate the formalism:

R1:
V	–	B	–	*	⇒
V	–	b	–	*	

R2:
B	–	B	–	*	⇒
b	–	b	–	*	

R3:
c	–		–	d	⇔
c	–	b	–	d	

R1 and R2 illustrate the iterative application of rules on strings: they sanction the lexical-surface strings $\langle$VBBB, Vbbb$\rangle$, where the second $\langle$B,b$\rangle$ pair serves as the centre of the first application of R2 and as the left context of the second application of the same rule. R3 is an epenthetic rule which also demonstrates centres of unequal length. (We assume that $\langle$V,V$\rangle$, $\langle$c,c$\rangle$ and $\langle$d,d$\rangle$ are sanctioned by other identity rules.)

The conditions in Definitions 2.1 and 2.2 that restrict the regular relations in the rules to being 'orthogonal' are required in order for the final language to be regular, because Definition 2.6 involves an implicit intersection of rule contexts, and we know that the intersection of regular relations is not in general regular.

2.2 Regular Expressions for Compilation

To compile a two-level grammar into an automaton we use a calculus of regular languages. We first use the standard technique of converting regular n-relations into same-length regular relations by padding them with a space symbol 0. Unlike arbitrary regular n-relations, same-length regular relations are closed under intersection and complementation, because a theorem tells us that they correspond to regular languages over (ϵ-free) n-tuples of symbols (Kaplan and Kay, 1994, p. 342).

A proposed morphological analysis $P = P_1 ... P_k$ can be represented as a same-length string-tuple $\omega \hat{P}_1 \omega \hat{P}_2 \omega ... \omega \hat{P}_k \omega$, where $\hat{P}_i \in \Sigma^*$ is P_i converted to a same-length string-tuple by padding with 0s, and $\omega = \langle \omega_1, ..., \omega_n \rangle$, where the $\{\omega_i\}$ are new symbols to indicate the partition boundaries, $\omega_i \notin \Sigma_i \cup \{0\}$.

Since in a partitioned string-tuple accepted by the grammar $(R_\Rightarrow, R_\Leftarrow)$ each $P_i \in c$ for some CR rule $(l, c, r) \in R_\Rightarrow$, we can make this representation unique by defining a canonical way of converting each such possible centre C into a same-length string-tuple $\hat{C}$. A simple way of doing this is to pad with 0s at the right making each string as long as the longest string in C: if $C = \langle p_1, ..., p_n \rangle$,

$$\hat{C} = \langle p_1 0^*, ..., p_n 0^* \rangle \cap \Sigma^* - \Sigma^* \langle 0, ..., 0 \rangle \qquad (1)$$

However, since we know the set of possible partitions – it is $\bigcup \{c \mid \exists l, r \langle l, c, r \rangle \in R_\Rightarrow\}$ – we can reduce the number of elements of Σ in use, and hence simplify the calculations, by inserting the 0s in a more flexible manner: e.g., if $C = \langle ab, b \rangle$, let $\hat{C} = \langle ab, 0b \rangle$ rather than $\hat{C} = \langle ab, b0 \rangle$: assuming

another rule requires us to use $\langle b, b\rangle$ anyway, we only have to add $\langle a, 0\rangle$ rather than $\langle a, b\rangle$ and $\langle b, 0\rangle$. The preprocessor could use simple heuristics to make such decisions. In any case, the padding of possible partitions carries over to the centres c of CR rules: if $(l, c, r) \in R_{\Rightarrow}$, $\hat{c} = \{\hat{C} \mid C \in c\}$. Henceforth let π be the set of elements of Σ that appear in some 0-padded rule centre.

The contexts of all rules and the lexical and surface centres of SC rules must be converted into same-length regular n-relations by inserting 0s at all possible positions on each tape independently: if $x = x_1 \times ... \times x_n$,

$$x^0 = (\texttt{Intro}_{\{0\}} x_1 \times ... \times \texttt{Intro}_{\{0\}} x_n) \cap \pi^* \quad (2)$$

Note the difference between this insertion of 0 everywhere, denoted x^0, and the canonical padding $\hat{c}$. Both require the 'orthogonality' condition in order for the intersection with π^* to yield a regular language: inserting 0s into $\langle a, b\rangle^*$ at all possible positions on each tape independently would give a non-regular relation, for example.

Now we derive a formula for the set of 0-padded and partitioned analysis strings accepted by the grammar $(R_{\Rightarrow}, R_{\Leftarrow})$: The set of 0-padded centres of context restriction rules is given by:

$$D = \{\hat{c} \mid \exists l, c, r.(l, c, r) \in R_{\Rightarrow}\} \quad (3)$$

Here we assume that these centres are disjoint ($\forall c, d \in D.c = d \vee c \cap d = \emptyset$), because in practice each c is a singleton set, however there is an alternative derivation that does not require this.

We proceed subtractively, starting as an initial approximation with an arbitrary concatenation of the possible partitions, i.e. the centres of CR rules:

$$\omega(D\omega)^* \quad (4)$$

From this we wish to subtract the set of strings containing a partition that is not allowed by any CR rule: We introduce a new placeholder symbol τ, $\tau \notin \pi \cup \{\omega\}$, to represent the centre of a rule, so the set of possible contexts for a given centre $\hat{c} \in D$ is given by:

$$\bigcup_{(l,\hat{c},r)\in R_{\Rightarrow}} l^0 \tau r^0 \quad (5)$$

So the set of contexts in which the centre c may *not* appear is the complement of this:

$$\pi^* \tau \pi^* - \bigcup_{(l,\hat{c},r)\in R_{\Rightarrow}} l^0 \tau r^0 \quad (6)$$

Now we can introduce the partition separator ω throughout, then substitute the centre itself, $\omega\hat{c}\omega$, for its placeholder τ in order to derive an expression for the set of partitioned strings in which an instance of the centre c appears in a context in which it is *not* allowed: [$\circ$ denotes composition]

$$\texttt{Sub}_{\omega\hat{c}\omega,\tau} \circ \texttt{Intro}_{\{\omega\}} \left(\pi^* \tau \pi^* - \bigcup_{(l,\hat{c},r)\in R_{\Rightarrow}} l^0 \tau r^0 \right) \quad (7)$$

If we subtract a term like this for each $\hat{c} \in D$ from our initial approximation (eq. 4), then we have an expression for the set of strings allowed by the CR rules of the grammar:

$$\omega(D\omega)^* \quad - \quad \bigcup_{\hat{c}\in D} \texttt{Sub}_{\omega\hat{c}\omega,\tau} \circ \texttt{Intro}_{\{\omega\}} \left(\pi^* \tau \pi^* - \bigcup_{(l,\hat{c},r)\in R_{\Rightarrow}} l^0 \tau r^0 \right) \quad (8)$$

It remains to enforce the surface coercion rules $R_{\Leftarrow}$. For a given SC rule $(l, c_l, c_s, r) \in R_{\Leftarrow}$, a first approximation to the set of strings in which this rule is violated is given by:

$$\texttt{Intro}_{\{\omega\}}(l^0 \omega (c_l^0 - c_s^0) \omega r^0) \quad (9)$$

Here $(c_l^0 - c_s^0)$ is the set of strings that match the lexical centre but do not match the surface centre. For part (2) of Definition 2.6 to apply this must equal the concatenation of 0 or more adjacent partitions, hence it has on each side of it the partition separator ω, and the operator `Intro` introduces additional partition separators into the contexts and the centre. The only case not yet covered is where the centre matches 0 adjacent partitions ($i = j$ in part (2) of Definition 2.6). This can be dealt with by prefixing with the substitution operator $\texttt{Sub}_{\omega,\omega\omega}$, so the set of strings in which one of the SC rules is violated is:

$$\bigcup_{(l,c_l,c_s,r)\in R_{\Leftarrow}} \texttt{Sub}_{\omega,\omega\omega} \circ \texttt{Intro}_{\{\omega\}}(l^0 \omega (c_l^0 - c_s^0) \omega r^0) \quad (10)$$

We subtract this too from our approximation (eq. 8) in order to arrive at a formula for the set of 0-padded and partitioned strings that are accepted by the grammar:

$$\begin{aligned} S_0 \quad = \quad & \omega(D\omega)^* - \bigcup_{\hat{c}\in D} \texttt{Sub}_{\omega\hat{c}\omega,\tau} \circ \\ & \texttt{Intro}_{\{\omega\}} \left(\pi^* \tau \pi^* - \bigcup_{(l,\hat{c},r)\in R_{\Rightarrow}} l^0 \omega' r^0 \right) \\ & - \bigcup_{(l,c_l,c_s,r)\in R_{\Leftarrow}} \texttt{Sub}_{\omega,\omega\omega} \circ \\ & \texttt{Intro}_{\{\omega\}}(l^0 \omega (c_l^0 - c_s^0) \omega r^0) \end{aligned} \quad (11)$$

Finally, we can replace the partition separator ω and the space symbol 0 by ϵ to convert S_0 into a regular (but no longer same-length) relation S that maps between lexical and surface representations, as in (Kaplan and Kay, 1994, p. 368).

3 Algorithm and Illustration

This section goes through the compilation of the sample grammar in section 2.1 step by step.

3.1 Preprocessing

Preprocessing involves making all expressions of equal-length. Let, $\Sigma_1 = \{$V,B,c,d,0$\}$ and $\Sigma_2 = \{$V,b,c,d,0$\}$ be the lexical and surface alphabets, respectively. We pad all centres with 0's (eq. 1), then compute the set of 0-padded centres (eq. 3),

$$D = \{\langle \text{B,b}\rangle, \langle \text{0,b}\rangle, \langle \text{V,V}\rangle, \langle \text{c,c}\rangle, \langle \text{d,d}\rangle\} \qquad (12)$$

We also compute contexts (eq. 2). Uninstantiated contexts become

$$\texttt{Intro}_{\{0\}}(\Sigma_1^*) \times \texttt{Intro}_{\{0\}}(\Sigma_2^*) \qquad (13)$$

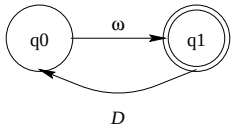
The right context of R3, for instance, becomes

$$\texttt{Intro}_{\{0\}}(\text{d}\Sigma_1^*) \times \texttt{Intro}_{\{0\}}(\text{d}\Sigma_2^*) \qquad (14)$$

3.2 Compilation into Automata

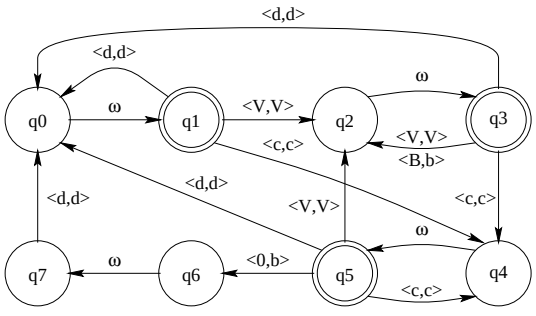
The algorithm consists of three phases: (1) constructing a FSA which accepts the centres, (2) applying CR rules, and (3) forcing SC constraints.

The first approximation to the grammar (eq. 4) produces FSA_1 which accepts all centres.



FSA_1

Phase 2 deals with CR rules. We have two centres to process: ⟨B,b⟩ (R1 & R2) and ⟨0,b⟩ (R3). For each centre, we compute the set of *invalid* contexts in which the centre occurs (eq. 7). Then we subtract this from FSA_1 (eq. 8), yielding FSA_2.



FSA_2

The third phase deals with SC rules: here the ⇐ portion of R3. Firstly, we compute the set of strings in which R3 is violated (eq. 10). Secondly, we subtract the result from FSA_2 (eq. 11), resulting in an automaton which only differs from FSA_2 in that the edge from q_5 to q_0 is deleted.

4 Comparison with Previous Compilations

This section points out the differences in compiling two-level rules in Koskenniemi's formalism on one hand, and the one presented here on the other.

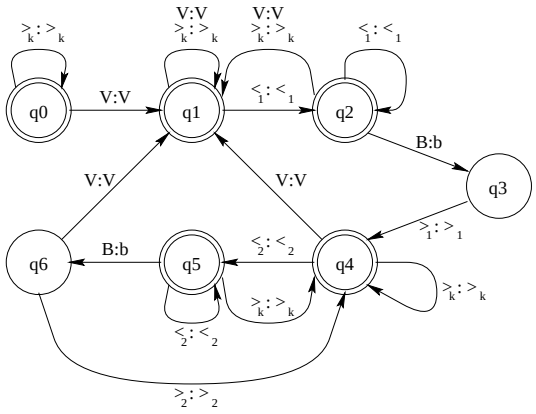
4.1 Overlapping Contexts

One of the most important requirements of two-level rules is allowing the multiple applications of a rule on the same string. It is this requirement which makes the compilation procedures in the Koskenniemi formalism – described in (Kaplan and Kay, 1994) – inconvenient. 'The multiple application of a given rule', they state, 'will turn out to be the major source of difficulty in expressing rewriting rules in terms of regular relations and finite-state transducers' (p. 346). The same difficulty applies to two-level rules.

Consider R1 and R2 (§2.1), and $D = \{\langle \text{V,V}\rangle, \langle \text{B,b}\rangle\}$. (Kaplan and Kay, 1994) express CR rules by the relation,[1]

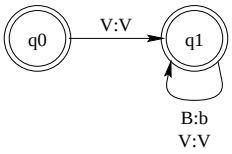
$$Restrict(c, l, r) = \overline{\overline{\pi^* l}\ c\pi^*} \ \cap\ \overline{\pi^* c\ \overline{r\pi^*}} \qquad (15)$$

This expression 'does not allow for the possibility that the context substring of one application might overlap with the centre and context portions of a preceding one' (p. 371). They resolve this by using auxiliary symbols: (1) They introduce left and right context brackets, $<_k$ and $>_k$, for each context pair $l_k - r_k$ of a specific centre which take the place of the contexts. (2) Then they ensure that each $<_k$:$<_k$ only occurs if its context l_k has occurred, and each $>_k$:$>_k$ only occurs if followed by its context r_k. The automaton which results after compiling the two rules is:



Removing all auxiliary symbols results in:

[1] This expression is an expansion of *Restrict* in (Kaplan and Kay, 1994, p. 371).



Our algorithm produces this machine directly. Compiling Koskenniemi's formalism is complicated by its interpretation: rules apply to the entire input. A partition rule is concerned only with the part of the input that matches its centre.

4.2 Conditional Compilation

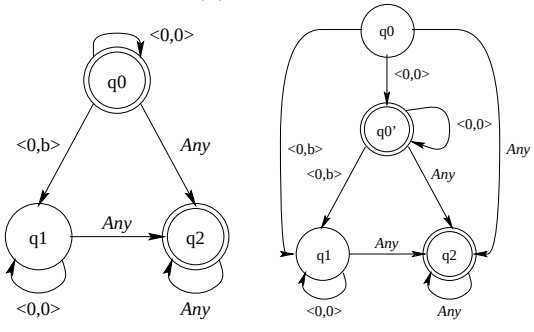
Compiling epenthetic rules in the Koskenniemi formalism requires special means; hence, the algorithm is conditional on the type of the rule (Kaplan and Kay, 1994, p. 374). This peculiarity, in the Koskenniemi formalism, is due to the dual interpretation of the `0` symbol in the parallel formalism: it is a genuine symbol in the alphabet, yet it acts as the empty string ϵ in two-level expressions. Note that it is the duty of the user to insert such symbols as appropriate (Karttunen and Beesley, 1992).

This duality does not hold in the partition formalism. The user can express lexical-surface pairs of unequal lengths. It is the duty of the rule compiler to ensure that all expressions are of equal length prior to compilation. With CR rules, this is done by padding zeros. With SC rules, however, the `Intro` operator accomplishes this task. There is a subtle, but important, difference here.

Consider rule R3 (§2.1). The 0-padded centre of the CR portion becomes ⟨0,b⟩. The SC portion, however, is computed by the expression

$$\texttt{Insert}_{\{0\}}(\epsilon) \times \overline{\texttt{Insert}_{\{0\}}(b)} \qquad (16)$$

yielding automaton (a):



a b

If the centre of the SC portion had been padded with 0's, the centre would have been

$$\texttt{Insert}_{\{0\}}(0) \times \overline{\texttt{Insert}_{\{0\}}(b)} \qquad (17)$$

yielding the undesired automaton (b). Both are similar except that state q_0 is final in the former. Taking (a) as the centre, eq. 10 includes ⟨cd,cd⟩; hence, eq. 11 excludes it. The compilation of our rules is not conditional; it is general enough to cope with all sorts of rules, epenthetic or not.

5 Conclusion and Future Work

This paper showed how to compile the partition formalism into N-tape automata. Apart from increased efficiency and portability of implementations, this result also enables us to more easily relate this formalism to others in the field, using the finite-state calculus to describe the relations implemented by the rule compiler.

A small-scale prototype of the algorithm has been implemented in Prolog. The rule compiler makes use of a finite-state calculus library which allows the user to compile regular expressions into automata. The regular expression language includes standard operators in addition to the operators defined here. The system has been tested with a number of hypothetical rule sets (to test the integrity of the algorithm) and linguistically motivated morphological grammars which make use of multiple tapes. Compiling realistic descriptions would need a more efficient implementation in a more suitable language such as C/C++.

Future work includes an extension to simulate a restricted form of unification between categories associated with rules and morphemes.

References

[Black et al., 1987] Black, A., Ritchie, G., Pulman, S., and Russell, G. (1987). Formalisms for morphographemic description. In *EACL-87*, pp. 11–8.

[Bowden and Kiraz, 1995] Bowden, T. and Kiraz, G. (1995). A morphographemic model for error correction in nonconcatenative strings. In *ACL-95*, pp. 24–30.

[Carter, 1995] Carter, D. (1995). Rapid development of morphological descriptions for full language processing systems. In *EACL-95*, pp. 202–9.

[Kaplan and Kay, 1994] Kaplan, R. and Kay, M. (1994). Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–78.

[Karttunen, 1994] Karttunen, L. (1994). Constructing lexical transducers. In *COLING-94*, pp. 406–411.

[Karttunen and Beesley, 1992] Karttunen, L. and Beesley, K. (1992). *Two-Level Rule Compiler*. Palo Alto Research Center, Xerox Corporation.

[Kiraz, 1994] Kiraz, G. (1994). Multi-tape two-level morphology: a case study in Semitic non-linear morphology. In *COLING-94*, pp. 180–6.

[Kiraz, 1996b] Kiraz, G. (1996b). *Computational Approach to Non-Linear Morphology*. PhD thesis, University of Cambridge.

[Koskenniemi, 1983] Koskenniemi, K. (1983). *Two-Level Morphology*. PhD thesis, University of Helsinki.

[Pulman, 1991] Pulman, S. (1991). Two level morphology. In Alshawi et. al, *ET6/1 Rule Formalism and Virtual Machine Design Study*, chapter 5. CEC, Luxembourg.

[Pulman and Hepple, 1993] Pulman, S. and Hepple, M. (1993). A feature-based formalism for two-level phonology: a description and implementation. *Computer Speech and Language*, 7:333–58.

[Ritchie et al., 1992] Ritchie, G., Black, A., Russell, G., and Pulman, S. (1992). *Computational Morphology: Practical Mechanisms for the English Lexicon*. MIT Press, Cambridge Mass.

[Ruessink, 1989] Ruessink, H. (1989). Two level formalisms. Technical Report 5, Utrecht Working Papers in NLP.

[Trost, 1990] Trost, H. (1990). The application of two-level morphology to non-concatenative German morphology. In Karlgren, H., editor, *COLING-90*, pages 371–6.